SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Laravel aplikacija za vođenje projekta

ZAVRŠNI RAD

Jure Bajić

Osijek, 2016

SADRŽAJ

1. UVOD

Zadatak ovog završnog rada je izrada *web* aplikacije za vođenje projekata u Laravel razvojnom okruženju (engl. *framework*). Pomoću aplikacije moguće je voditi više projekata istovremeno, te svaki projekt ima svog voditelja, članove koji sudjeluju u izradi projekta, skupine zadataka i zadatke. Voditelj projekta dodjeljuje zadatke članovima i određuje rokove. Članovi mogu vidjeti i izvršiti svoje zadatke. Samo voditelj projekta može zatvoriti projekt, promijeniti stavke projekta ili ga izbrisati. Izuzev osnovnih dijelova za vođenje projekta, nužno je da aplikacija ima sustav registracije, autentifikacije i dodavanja novih prijatelja kao potencijalnih članova projekta.

1.1. Zadatak završnog rada

Potrebno je izraditi *web* aplikaciju koja će korisniku omogućiti vođenje projekata, te povezivanje s ostalim korisnicima aplikacije. Pritom je potrebno izraditi bazu podataka, uspostaviti potrebne upite nad bazom, prikazati sadržaj korisniku i omogućiti mu upravljanjem istim.

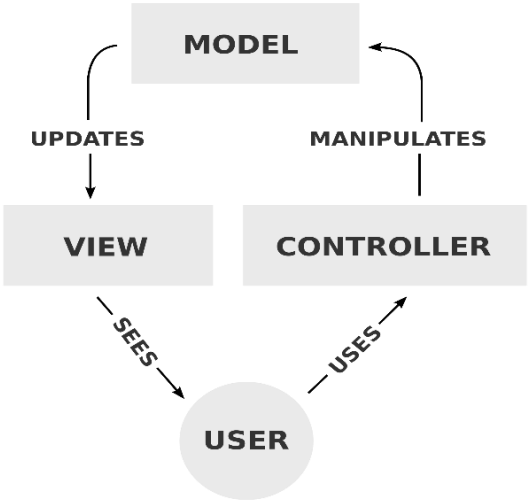
2. TEHNOLOGIJE

Sve korištene tehnologije su besplatne, te obuhvaćaju područje *back-enda* i *front-enda.* Korištene tehnologije su:

- Laravel 5.2
- Xampp s inačicom PHP-a 5.6.31. i MySQL-a 5.6.21
- HTML5
- CSS
- JavaScript; jQuery 2.2.3
- Bootstrap 3.3.6

2.1. Laravel

Prema literaturi [1], [2], Laravel je besplatno, *open-source* razvojno okruženje PHP-a čija osnova leži na već postojećem razvojnom okruženju Simfony 2. Laravel je razvio Taylor Otwell kako bi konkurirao već brojnim postojećim razvojnim okruženjima kao npr. Codeigniter, Zend, Slim itd. Prema literaturi [3], [4], Laravel se temelji na MVC (engl. *Model View Controller*) arhitekturi, koja podrazumijeva razdvajanje same logike aplikacije, pravila i upita nad bazom za koje je sve odgovoran model (engl. *model*), dok kontroler (engl. *controller*) prima korisnikove upite i prosljeđuje ih modelu, a pogled (engl. *view*) samo prikazuje podatke koje mu je kontroler proslijedio.



Sl. 2.1. *Prikaz MVC paradigme, [5]*

Cijeli kod Laravela je objavljen na stranicama GitHuba i licenciran je pod MIT licencom. Svatko može doprinijeti razvoju i poboljšanju koda. Laravel je koncipiran da koristi brojne druge dodatke ili pakete (engl. *dependency*), te se kao takav instalira uz jedan od programa za upravljanje paketima, u ovom slučaju koristio se Composer.

Dolaskom Laravela 3 uveo se CLI (engl. *Command line interface*) nazvan Artisan kojim se uvodi mogućnost lakšeg upravljanja bazom podataka putem migracija, automatskog dodavanja klasa s odgovarajućim uzorkom.

Brojni paketi koji su već uključeni u Laravel kojih ima preko 20, SwiftMailer, Blade, Carbon, Doctrine te brojne druge iz Symfonya kao npr. ClassLoader, Guard, Routing. Upravo modularnost čini Laravelom jednom od najboljih razvojnih okruženja, budući da iskorištava postojeće popularne i kvalitetno napisane komponente te ih objedinjuje u cjelinu u kojoj korisnik može vrlo lagano ažurirati svaki modul zasebno pomoću Composera.

Za kreiranje baze podataka te njenu konzistentnu izmjenu postoje migracije (engl. *migrations*) kojima se definira struktura baze podataka, a istu je moguće popuniti podacima s *seedsovima.*

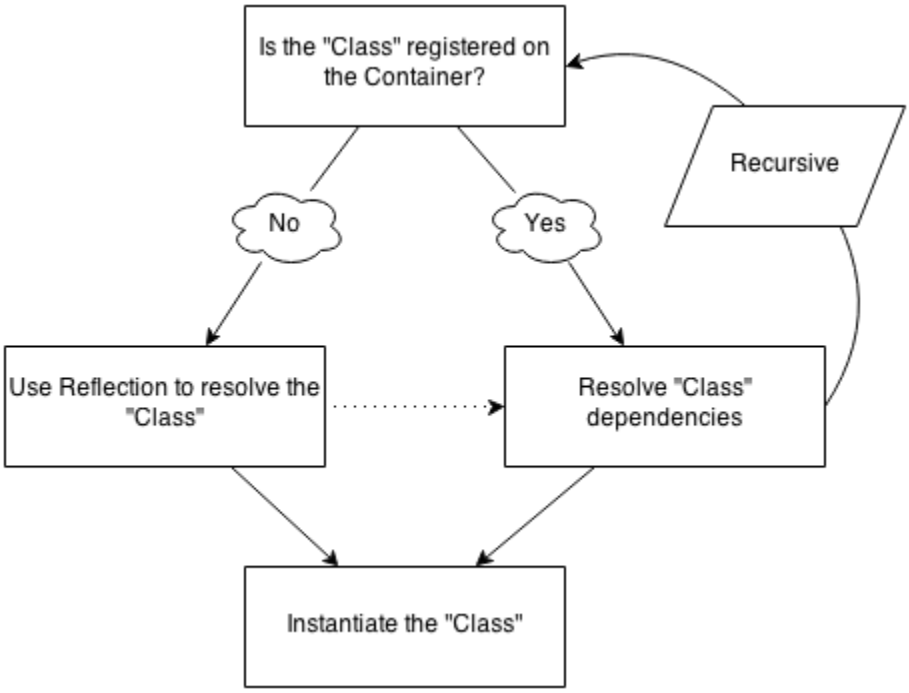
Rad s bazom podataka omogućuje *query builder* koji zamjenjuje pisanje upita nad bazom nizanjem metoda, isto je moguće napraviti pomoću ORM-a (engl. *Object relational mapping*) zvan Eloquent. Svaki model predstavlja entitet u bazi, te kao takav ima atribute koji odgovaraju atributima u bazi, a metodama su prikazane veze s ostalim entitetima. Eloquent koristi ActiveRecord čime se rad s bazom olakšava tako što se definiranjem svojstava objekata definiraju i atributi zapisa u bazi podataka.

```
$user = new User;

$user->username = 'luka';

$user->email = 'luka@mail.com';

$user->password = '123456';

$user->save();
```

Sl. 2.2. *Prikaz stvaranja zapisa u bazu putem objekta ($user) klase odnosno modela Usera, [1]*

Prema literaturi [6], Laravelov IoC (engl. *Inversion of Control*) ili inverzija kontrole je vrsta dizajna aplikacije u kojoj za razliku od tradicionalnog programiranja u kojem korisnikove naredbe

pozivaju naredbe koje se nalaze u razvojnom okruženju, samo razvojno okruženje poziva korisnikove naredbe. Takav način arhitekture povećava modularnost aplikacije i čini ju lako prilagodljivom novim modulima.

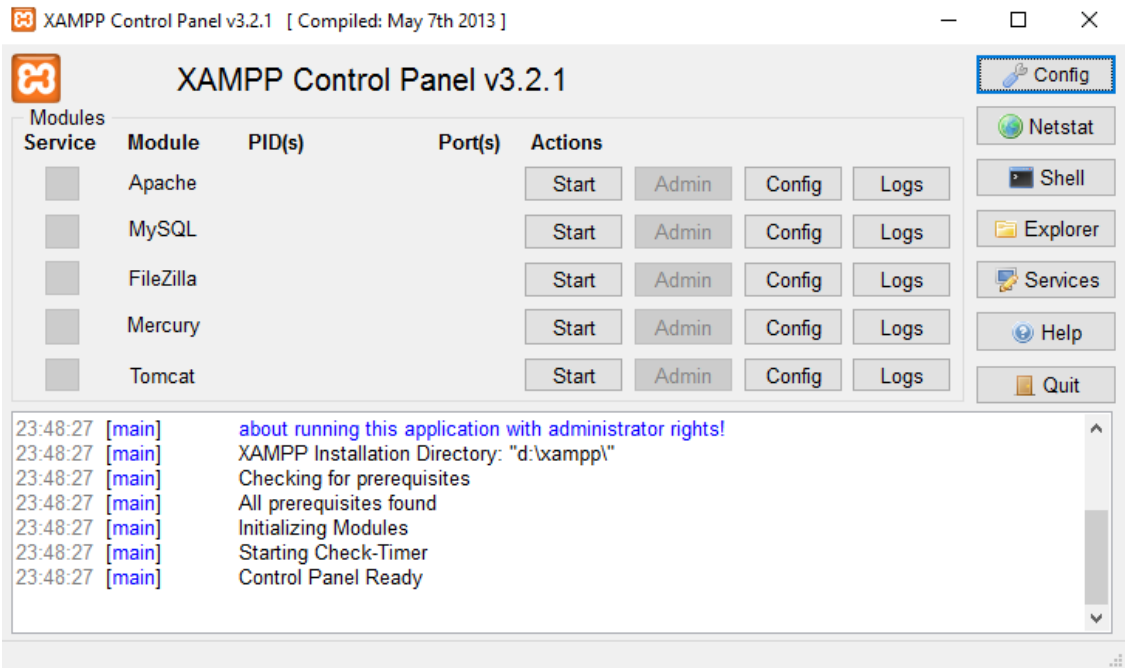


Sl. 2.3. *Shema rada Laravelovog IoC, [6]*

Unutar Laravela nalazi se i paket koji omogućuje lagano postavljanje obrasca pogleda zvan Blade.

2.2. XAMPP

XAMPP je open-source multiplatforma s kojom dolazi server Apache, PHP, MySQL, MariaDB i brojne druge aplikacije. Kao takav je pogodan za razvijanje i testiranje *web* aplikacija na lokalnom računalu.



Sl. 2.4. *Prikaz kontrolne ploče XAMPP-a gdje se mogu pokrenuti HTTP server Apache i MySQL baza podataka, [7]*

2.2.1. PHP

Prema literaturi [8], PHP (engl. *Hypertext Preprocessor*) je skriptni jezik isključivo namijenjen za izradu dinamičkih *web* stranica odnosno aplikacija. PHP je najrasprostranjeniji jezik za izradu *web* aplikacija te je zbog toga prisutan na skoro svim *web* serverima te se može implementirati na skoro svim operacijskim sustavima. Standardni PHP interpreter je Zend engine koji je također kao i PHP besplatan.

2.2.2. MySQL

MySQL je *open-source* relacijski sustav za vođenje baze podataka (engl. RDBMS *relational database management system*). Jezik korišten za definiranje strukture baze podataka i manipuliranje nad bazom podataka je SQL.

2.3. HTML5

Prema literaturi [9], [10], HTML5 (engl. *HyperText Markup Language*) je jezik kojim se opisuje izgled *web* stranica. HTML je neizostavan element izrade bilo kakve *web* stranice, *web* aplikacije i mobilnih aplikacija. Intrepretira ga preglednik, a sam HTML nije programski jezik već se njime opisuje sadržaj stranice pomoću elemenata, tzv. tagovima *(engl. tags)* koji označavaju početak i kraj elementa, npr. `<div></div>`. HTML5 je nova inačica HTML s dodanim brojnim novim elementima.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
</body>
</html>
```

Sl. 2.5. *Prikaz osnovnih elemenata svakog HTML dokumenta, [8]*

2.4. CSS

CSS (engl. *Cascading style sheets*) je jezik za dodatno opisivanje odnosno stiliziranje HTML elemenata te određivanje načina na koji će se prikazati. Svrha CSS jest odvajanje sadržaja dokumenta od samog načina prezentiranja dokumenta korisniku. Prema literaturi [7], nastao je zbog pojave gomile HTML elemenata koji su služili stiliziranju izgleda stranice i time uzrokovali nečitljivu i teško održivu strukturu koda, te je W3C (engl. *World Wide Web Consortium*) odlučio stvoriti CSS kako bi se razdvojio sadržaj dokumenta od prezentiranja istog.

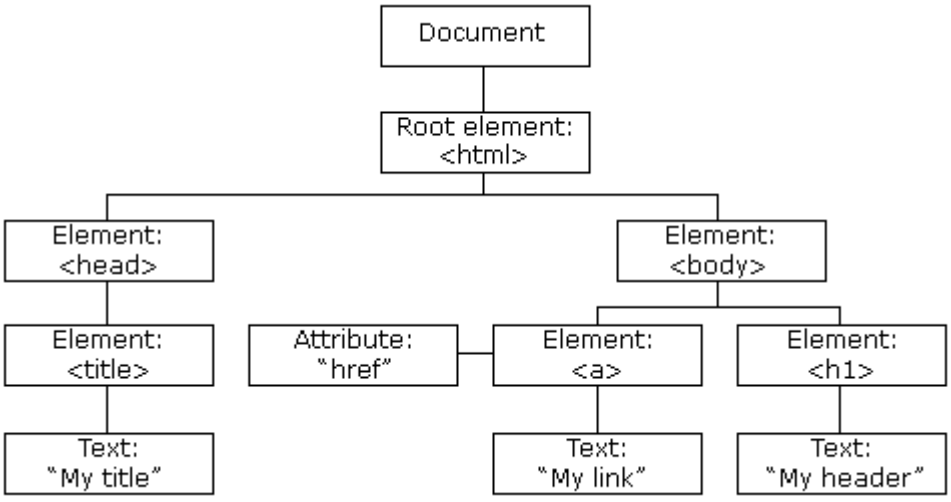
2.5. JavaScript

JavaScript je skriptni jezik koji *web* aplikacijama daje dinamičnost te daje mogućnost stvaranja interaktivnost s korisnikove strane, te zajedno s HTML-om i CSS-om čini osnovnu tehnologija za

razvijanje internetskih sadržaja. Za razliku od HTML-a i CSS-a, JavaScript se počeo koristiti i za razvijanje serverske logike web aplikacija *Node.js*. Standard JavaScripta je standardiziran *ECMAScript* specifikacijama.

2.5.1. jQuery

Prema literaturi [11], JQuery je JavaScript biblioteka koja olakšava manipulaciju elementima HTML-a. Iako je samo biblioteka jQuery je uveo velike promjene u sam način pisanja skripti, a sintaksa je ostala nepromijenjenom. Kako bi JavaScript mogao prolaziti kroz HTML dokument te upravljati samom strukturom tog dokumenta uveden je DOM (engl. *Document Object Model*), koji se definira kao API (engl. *Application programming interface*) koje tretira HTML, XML ili XHTML kao stablastu strukturu.



Sl. 2.6. *Prikaz stablaste strukture DOM-a, [8]*

2.6. Bootstrap

Prema literaturi [12], Bootstrap je *open-source front-end* razvojno okruženje. Sadrži HTML i CSS obrazac i već predefinirane klase za HTML elemente. Također sadrži i JavaScript ekstenziju.

Mark Otto i Jacob Thorton su razvili Twitter Blueprint 2011. godine koji se kasnije preimenovao u Bootstrap.

Bootstrap je modularni tip razvojnog okruženja što znači da se sastoji od skupa *stylesheetsova* koji implementiraju različite komponente u okružje. Osoba koja razvija u tom razvojnom okruženju ima mogućnost izbora komponenti po potrebi, te također može iste dokumente prilagoditi svojim potrebama.

3. BAZA PODATAKA

Svaka *web* aplikacija u svojoj pozadini mora imati bazu podataka kako bi zapisi ostali sačuvani i nakon odlaska korisnika s iste. Baza podataka održava konzistentnost podataka koje se unose to omogućuje povezivanje podataka u nužne relacije i time olakšava i ubrzava postupak dohvaćanja podataka. Kako bi se baza podataka uspješno kreirala potrebno je prvo postaviti uvjete i zahtjeve koje mora zadovoljiti. Način izrade baze podataka rađen je po literaturi [13].

3.1. Utvrđivanje i analiza zahtjeva

Potrebno je napraviti bazu podataka za web aplikaciju koja svojim korisnicima omogućuje međusobno povezivanje te vođenje projekata. Svaki korisnik ima skup drugih korisnika s kojima je povezan, te može imati jedan ili više projekata na kojima trenutno radi, a na projektu može raditi jedan ili više korisnika. Korisnik ako je voljan može ispuniti svoj profil i odabrati zemlju u kojoj se nalazi. Svaki projekt ima voditelja projekta koji je zadužen za upravljanje stavkama projekta. Svaki projekt ima i određeni broj skupina zadataka pa tako i zadataka, a svaki zadatak je povezan s skupinom izvršitelja zadatka.

Entiteti: USERS, PROJECTS, TASKS, COUNTRIES

Veze: PRIJATELJI između USERS i USERS, SUDJELUJU između USERS i PROJECTS, UPRAVLJA između USERS i PROJECTS, RADE između USERS i TASKS, PRIPADA između PROJECT i TASK, PRIPADA između TASKS i TASKS, IZ između USERS i COUNTRIES

Entitet USERS ima atribute: id, first_name, last_name, username, email, password, body, address, city, country_id, avatar, created_at, updated_at, deleted_at, remember_token.

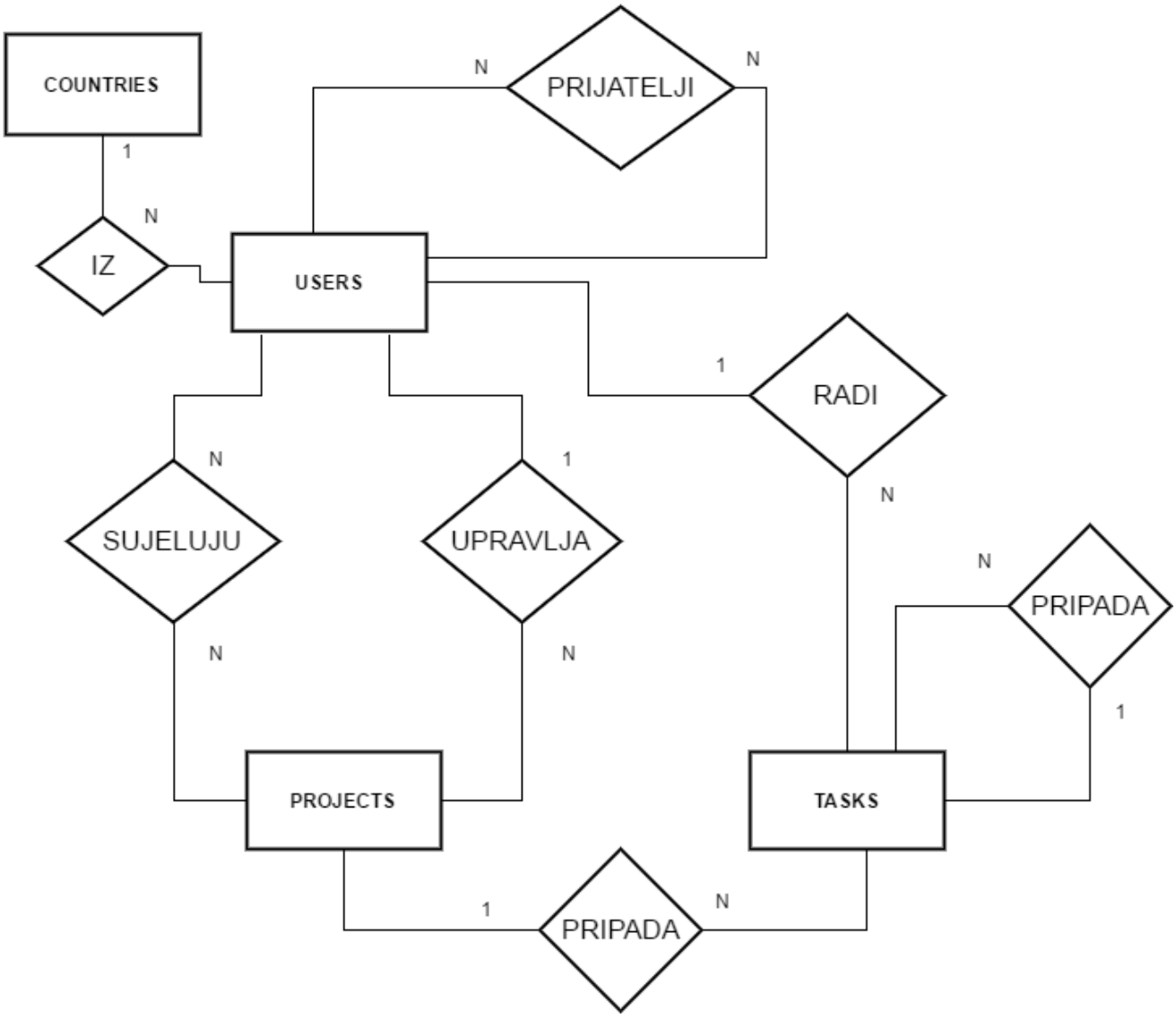
Entitet PROJECTS ima atribute: id, name, body, deadline, client, completed, created_at, updated_at, manager_id.

Entitet TASKS ima atribute: id, name, body, completed, estimated_time, deadline, project_id, task_id, created_at, updated_at.

Entitet COUNTRIES ima atribute: id, name, code.

3.2. Konceptualno oblikovanje

Na temelju analize zahtjeva baze podataka dobiven je ER model prikazan Chenovim dijagramom. Entiteti se nalaze u pravokutnicima, a prikazane veze su jedan na više (1:N) i više na više (N:N).



Sl. 3.1. *ER model baze podataka*

Slika 3.1. prikazuje ER (engl. *entity relationship*) model baze podataka gdje je vidljiva sama struktura entiteta baze podataka i njihovih međusobnih relacija. Takav dijagram nam daje grafički prikaz strukture baze podataka i relacija unutar baze.

3.3. Logičko oblikovanje

Kako bismo mogli fizički oblikovati bazu podataka potrebno je transformirati ER model u relacijski model u kojem se entitet i veza iz ER modela pretvara u relaciju. Relacijska shema baze podataka prikazana je na slici 3.2.

COUNTRIES (id, name, code)

USERS (id, first_name, last_name, username, email, password, body, address, city, country_id, created_at, updated_at, deleted_at, remember_token)

PROJECTS (id, name, body, deadline, client, completed, created_at, updated_at, manager_id)

TASKS (id, name, body, completed, estimated_time, deadline, project_id, task_id, created_at, updated_at)

PROJECT_USER (project_id, user_id)

TASK_USER (task_id, user_id)

Sl. 3.2. *Prikaz relacijskog modela baze podataka.*

Tablica 3.1. prikazuje popis svih atributa, njihovih tipova i opis odnosno njihova uloga u prikazivanju određene relacije u bazi.

Tabl. 3.1. *Prikaz svih atributa, njihovih tipova i njihovog značenja u bazi podataka*

ATRIBUT	TIP	OPIS
id	UNSIGNED INT Cijeli broj u rasponu 0 – 4 294 967 295	Jedinstveni broj koji označava svaku n-torku
first_name	VARCHAR (30) – skup od maksimalno 30 znakova	Ime korisnika
last_name	VARCHAR (30) – skup od maksimalno 30 znakova	Prezime korisnika
username	VARCHAR (30) – skup od maksimalno 30 znakova	Korisničko ime korisnika
email	VARCHAR (255) – skup od maksimalno 255 znakova	Email korisnika
password	VARCHAR (60) – skup od maksimalno 60 znakova	Korisnikova zaporka koja se koristi prilikom autentifikacije
body	TEXT – skup od maksimalno 65 535 znakova	Dodatan opis određene relacije
address	VARCHAR (60) – skup od maksimalno 60 znakova	Korisnikova adresa
city	VARCHAR (60) – skup od maksimalno 60 znakova	Korisnikov grad stanovanja
name	VARCHAR (60) – skup od maksimalno 60 znakova	Naziv određene relacije
avatar	VARCHAR (255) – skup od maksimalno 255 znakova	Naziv slike koja predstavlja korisnikov avatar
deadline	DATE – datum formata YYY-MM-DD	Datum postavljenog završetka projekta ili zadatka
completed	TINYINT(1) -cijeli broj koji može poprimit stanje 0 ili 1	Označava status projekta
estimated_time	UNSIGNED INT Cijeli broj u rasponu 0 – 4 294 967 295	Označava potrebno vrijeme u satima za završetak zadatka
code	CHAR(2) – fiksan skup znakova kojih mora biti 2	Označava kratice određene države
created_at	TIMESTAMP – precizna vremenska oznaka formata YYYY-MM-DD HH:MM:SS	Vrijeme kreiranja n-torke
updated_at	TIMESTAMP – precizna vremenska oznaka formata YYYY-MM-DD HH:MM:SS	Vrijeme izmjene n-torke
deleted_at	TIMESTAMP – precizna vremenska oznaka formata YYYY-MM-DD HH:MM:SS	Vrijeme brisanja n-torke

3.4. Fizičko oblikovanje

Nakon oblikovanja baze podataka nužno je istu i fizički implementirati. Bazu podataka se može definirati na dva načina putem SQL naredbi prema literaturi [15], ili Laravelovih migracija pisane PHP-ovom sintaksom. Slika 3.3. prikazuje skup SQL naredbi nužnih za kreiranje baze podataka u MySQL relacijskom sustavu za vođenje baze podataka. Pri samom stvaranju baze podataka potrebno je definirati skup znakova koje će baza koristiti prilikom spremanja i iščitavanja zapisa, u ovom slučaju postavljen je UTF-8 (engl. *charset*), dok *collation* predstavlja skup pravila kojima će se ti znakovi uspoređivati te je postavljen na utf8_unicode_ci.

```
DROP DATABASE IF EXISTS projectmanagement;

CREATE DATABASE projectmanagement CHARACTER SET utf8 COLLATE
utf8_unicode_ci;
USE projectmanagement;

CREATE TABLE users(
      id                INT UNSIGNED NOT NULL AUTO_INCREMENT,
      first_name        VARCHAR(30),
      last_name         VARCHAR(30),
      username          VARCHAR(30) NOT NULL,
      email             VARCHAR(255) NOT NULL UNIQUE,
      password          VARCHAR(60),
      body              TEXT,
      address           VARCHAR(60),
      city              VARCHAR(60),
      avatar                  VARCHAR(255),
      country_id        INT UNSIGNED,
      created_at        TIMESTAMP,
      updated_at        TIMESTAMP,
      deleted_at        TIMESTAMP,
      remember_token    VARCHAR(100),
      CONSTRAINT PK_clanovi PRIMARY KEY(id)
);

CREATE TABLE projects(
      id                INT UNSIGNED NOT NULL AUTO_INCREMENT,
      name              VARCHAR(60),
      body              TEXT,
      deadline          DATE,
      client                  VARCHAR(60),
      completed         TINYINT(1) DEFAULT 0,
      created_at        TIMESTAMP,
      updated_at        TIMESTAMP,
      manager_id        INT UNSIGNED,
      CONSTRAINT PK_projects PRIMARY KEY(id)
);

CREATE TABLE project_user(
      project_id        INT UNSIGNED,
      user_id           INT UNSIGNED
);
```

```
CREATE TABLE tasks(
      id                INT UNSIGNED NOT NULL AUTO_INCREMENT,
      name              VARCHAR(60),
      body              VARCHAR(255),
      completed         TINYINT(1) DEFAULT 0,
      estimated_time    INT UNSIGNED,
      deadline          DATE,
      project_id        INT UNSIGNED,
      task_id           INT UNSIGNED,
      created_at        TIMESTAMP,
      updated_at        TIMESTAMP,
      CONSTRAINT PK_tasks PRIMARY KEY(id)
);

CREATE TABLE task_user(
      task_id           INT UNSIGNED,
      user_id           INT UNSIGNED
);

CREATE TABLE countries(
      id                INT UNSIGNED NOT NULL AUTO_INCREMENT,
      name              VARCHAR(60),
      code              CHAR(2),
      CONSTRAINT PK_countries PRIMARY KEY(id)
);

ALTER TABLE users ADD CONSTRAINT FK_users_countries FOREIGN KEY
(country_id) REFERENCES countries (id);

ALTER TABLE projects ADD CONSTRAINT FK_projects_users FOREIGN KEY
(manager_id) REFERENCES users (id);

ALTER TABLE project_user ADD CONSTRAINT FK_project_user_projects FOREIGN
KEY (project_id) REFERENCES projects (id);
ALTER TABLE project_user ADD CONSTRAINT FK_project_user_users FOREIGN KEY
(user_id) REFERENCES users (id);

ALTER TABLE tasks ADD CONSTRAINT FK_tasks_projects FOREIGN KEY (project_id)
REFERENCES projects (id);
ALTER TABLE tasks ADD CONSTRAINT FK_tasks FOREIGN KEY (task_id) REFERENCES
tasks (id);

ALTER TABLE task_user ADD CONSTRAINT FK_task_user_tasks FOREIGN KEY
(task_id) REFERENCES tasks (id);
ALTER TABLE task_user ADD CONSTRAINT FK_task_user_users FOREIGN KEY
(user_id) REFERENCES users (id);
```

Sl. 3.3. *Prikaz SQL koda za kreiranje baze podataka „projectmanagement“*

Prema slici 3.3., gdje je prikazan kod za kreiranje baze podataka, vidljiv je naziv baze podataka „projectmanagement“ koji je proizvoljan. Slika prikazuje kod za kreiranje baze podataka i njezinih pripadajućih tablica.

Kao što je već spomenuto postoji i drugi način za kreiranje baze podataka, a to je putem migracija koje se nalaze zajedno s ostatkom koda aplikacije u prilogu P.1.

```
Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('first_name', 30);
            $table->string('last_name', 30);
            $table->string('username', 30);
            $table->string('email')->unique();
            $table->string('password', 60);
            $table->string('address', 60);
            $table->string('city', 60);
            $table->string('avatar')->default('no-avatar.png');
            $table->text('body');
            $table->string('confirmation_code', 30)->nullable();
            $table->boolean('activated')->default(false);
            $table->rememberToken();
            $table->timestamps();
            $table->softDeletes();
        });
```

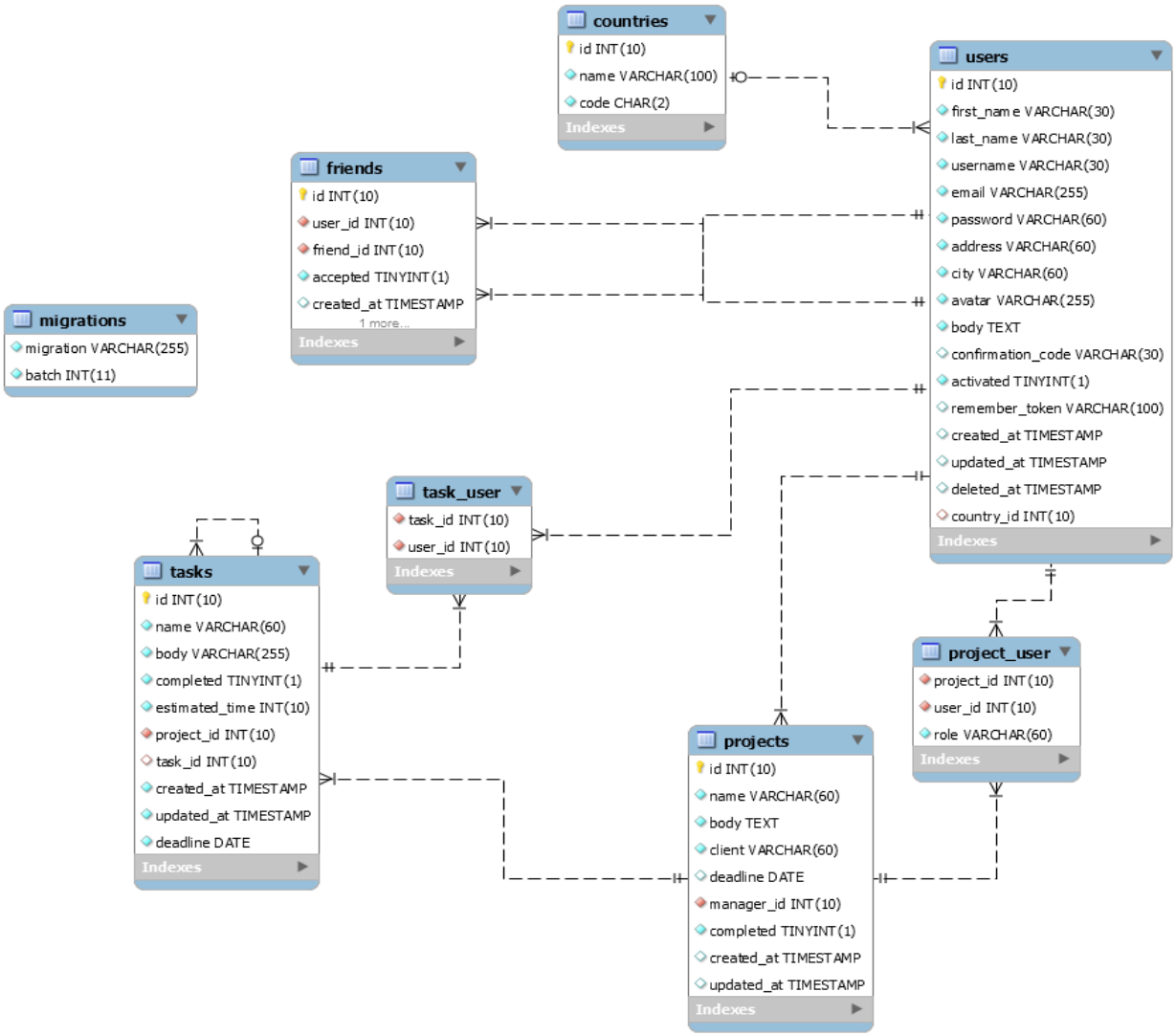
Sl. 3.4. *Isječak koda dokumenta ./database/migrations/2016_04_10_142812_create_users_table.php*

Slika 3.4. prikazuje PHP-ovu sintaksu za kreiranje tablice odnosno relacije USERS korištenjem Laravelove klase Schema te metode *create*. Kako bi migracije mogle uspješno raditi nužno je podesiti sljedeće parametre DB_HOST, DB_DATABASE, DB_USERNAME i DB_PASSWORD potrebne za spajanje s bazom podataka u dokumentu *./.env*. Svakako prije toga potrebno je kreirati samu bazu putem aplikacije phpMyAdmin ili putem komandne linije (engl. *command-line*). Navedeni parametri se koriste u datoteci *./config/database.php* prilikom uspostavljanja veze, te u toj datoteci možemo postaviti dodatne parametre kao npr. način spajanja s bazom podataka PDO ili MySQLi. Nakon što su svi parametri postavljeni i baza podataka je kreirana, jednostavnom komandom „php artisan migrate“ pokreću se sve migracije i kreiraju tablice.

Izuzev migracija, Laravel nudi mogućnost popunjavanja tablica testnim podacima putem *seedsova*. Primjer takvih datoteka se može vidjeti u prilogu P.1. *./database/seeds/*.

Nakon što je baza podataka kreirana postoji mogućnost provjere ispravnosti strukture baze podataka te veza među relacijama. Iskoristit ćemo besplatnu aplikaciju MySQL Workbench koja nam daje grafički prikaz stvorene baze podataka, također se može koristit i XAMPP-ova aplikacija phpMyAdmin koja isto daje mogućnost grafičkog prikaza, ali MySQL Workbench daje pregledniju sliku. Kako bi se dobio grafički prikaz baze podataka koristit će se opcija *reverse engineering*. Prvo je potrebno uspostaviti vezu s bazom podataka postavljanjem parametara

Connection name ime veze koje je proizvoljno, *host* naziv servera koji je uobičajeno lokalni prema tome 127.0.0.1, *port* na kojem se nalazi baza podataka koji je uobičajeni 3306 te korisničko ime i zaporku za pristup sustavu za upravljanje bazom podataka. Nakon što se veza uspješno uspostavi u alatnoj traci se odabere *Database* → *Reverse engineering*, te ispunimo tražene parametre i kao rezultat dobijemo grafički prikaz baze podataka prikazan na slici Sl. 3.5.



Sl 3.5. *Grafički prikaz baze podataka dobiven putem programa MySQL Workbencha*

4. IZRADA APLIKACIJE

Izrada aplikacije podijeljena je u tri segmenta; modeli, kontroleri i pogledi, prema MVC arhitekturi. Sama izrada aplikacije nije se odvijala tako, prilikom izrade sva tri segmenta su se paralelno razvijala budući da funkcionalno i jesu povezani, za pomoć prilikom izrade aplikacije koristila se [15] literatura. Za praćenje napretka i mogućnosti obnove stare inačice aplikacije u slučaje pogreške korišten je Git kao program kontrole inačice aplikacije, a cijeli kod je objavljen javno na stranicama GitHuba u literaturi [16]. Na danoj stranici moguće je vidjeti sve promjene i inačice u *master* grani i također preuzeti kod za vlastite potrebe.

4.1. Modeli

Modeli su predstavnici relacija u bazi podataka te se sva logika s bazom podataka odvija posredstvom modela unutar kojih se nalaze i pravila potvrđivanja (engl. *validation*) za utvrđivanje pravilnosti korisnikovih podataka i ostali potrebni atributi i metode za pravilno upravljanje klasom modela. Zadatak modela je dohvaćanje podataka iz baze podataka i spremanje u odgovarajuće objekte. Svi se modeli nalaze u direktoriju *./app/Models/* i svi nasljeđuju Laravelovu klasu *Model.* Specifičan je *User* model koji nasljeđuje klasu *Authenticatable,* klasa namijenjena isključivo modelu koji se koristi za autentifikaciju. Metode modela mogu biti različitog tipa ovisno o tome koja im je svrha te prema tome razlikujemo: *relationships*, *mutators*, *accessors i scopes*.

Relationships su metode koje definiraju vezu modela s drugim modelima, veze u kojima model može biti su jedan na jedan, jedan na više i više na više i njima sukladne funkcije. Funkcija *hasOne* koristi se u vezama jedan na jedan u jednom modelu, dok u drugom definiramo obrnutu vezu funkcijom *belongsTo.* Funkcija *belongsTo* se također koristi u vezama jedan na više kako bi se definirala veza modela koji pripada drugom modelu, dok se u drugom modelu definira obrnuta veza funkcijom *hasMany*. I zadnja vrsta veze više na više koja se definira u oba modela istom funkcijom *belongsToMany.*

Mutators služe kako bi se atributi izmijenili prije spremanja u bazu podataka. Definiraju se sintaksom *setFooAttribute* gdje je *Foo* proizvoljan naziv atributa koji se mijenja.

Accessors su metode koje izmjenjuju podatke nakon dohvata iz baze podataka. Definiraju se sintaksom *getFooAttribute* gdje je *Foo* proizvoljan naziv atributa koji se mijenja.

Scopes su metode kojima postavljamo određene uvjete za upite nad bazom podataka. Pišu se sintaksom *scopeFoo* gdje je *Foo* proizvoljan naziv kojim ih i pozivamo.

4.1.1. User

Prvi stvoreni model putem Laravelovog Artisana naredbom prikazanoj na slici 4.1. je model *User* koji se odnosi na tablicu *users*. Kao što je već rečeno taj model koristi se za autentifikaciju korisnika što je postavljeno u konfiguracijskom dokumentu *./config/auth.php.*

```
Jure@PISI /D/Xampp/htdocs/ProjectManagement (master)
$ php artisan make:model User
```

Sl. 4.1. *Naredba za stvaranje modela Usera čije sintaksa se može upotrijebiti za stvaranje i ostalih modela promjenom naziva (User)*

Veze koje su postavljene prilikom izrade baze podataka postavljaju se i modelu *User* prikazano na slici 4.2.

```
    public function tasks()
    {
        return $this->belongsToMany('App\Models\Task', 'task_user',
                            'user_id', 'task_id');
    }

    public function projects()
    {
        return $this->belongsToMany('App\Models\Project', 'project_user',
                            'user_id', 'project_id');
    }

    public function country()
    {
        return $this->belongsTo('App\Models\Country', 'country_id', 'id');
    }

    public function managerTo()
    {
```

Sl. 4.2. *Isječak koda iz dokumenta ./app/Models/User.php prikazuje metode potrebne za vezu s ostalim modelima, P.1.*

Iz slike 4.2. možemo jasno iščitati na koji način je model *User* povezan s ostalim modelima. Metoda *tasks* kazuje kako objekt modela *User* pripada više objekata modela *Task,* a koristi se za dobivanje svih korisnika zaduženih za izvršavanje određenog zadatka. Metoda *projects* kazuje kako objekt model *User* pripada više objekata modela *Project,* a koristi se za dobivanje svih projekata u kojima korisnik sudjeluje. Metoda *country* kazuje kako objekt modela *User* pripada jednom objektu modela *Country,* a koristi se za dobivanje zemlje u kojoj se korisnik nalazi. Metoda *managetTo* kazuje kako objekt modela *User* ima više objekata modela *Project,* a koristi se za dobivanje svih projekata čiji voditelj je korisnik.

Unutar *User* modela nalaze se atributi *login_rules, registration_rules* i *edit_rules* koji sadrže pravila za validaciju korisnikovih podataka. Primjer *registration_rules* varijable koja ima svojstvo statičnosti (engl. *static*) što znači da ju možemo pozvati bez da smo prethodno instancirali objekt, prikazan je na slici 4.3., a koristi se prilikom registracije korisnika. Skup svih mogućih parametara koji se mogu koristiti za definiranje željenih pravila validacije mogu se pronaći u Laravelovoj dokumentaciji.

```
public static $registration_rules = [
        'email' => 'required|email|unique:users',
        'password' => 'required|min:6|confirmed',
    ];
```

Sl. 4.3. *Pravila za provjeru ispravnosti formata danih podataka, P.1.*

Model *User* sadrži i metode *accessora* i *mutatorsa,* odnosno *getNameAttribute* i *setPasswordAttribute. accessor* getNameAttribute poziva se nakon što se objekt modela *User* instancira te se postavlja atribut *name,* dok se *mutator setPasswordAttribute* poziva prilikom spremanja korisnika u bazu podataka i tom se metodom zaporka korisnika kriptira radi sigurnosti i zaštite podataka korisnika. Ostale metode koje model sadrži koriste se kako bi se korisnici povezali putem relacije *friends,* te se mogu detaljnije pogledati u samom modelu.

4.1.2. Project

Model *Project* predstavlja relaciju *projects* što je definirano atributom *table*. Na slici 4.4. prikazane su metode koje definiraju veze modela *Project* s ostalim modelima.

```
    public function users()
    {
      return $this->belongsToMany('App\Models\User', 'project_user',
                                  'project_id', 'user_id');
    }

    public function tasks()
    {
      return $this->hasMany('App\Models\Task', 'project_id', 'id');
    }

    public function manager()
    {
        return $this->belongsTo('App\Models\User', 'manager_id', 'id');
    }
```

Sl. 4.4. *Isječak koda iz dokumenta ./app/Models/Project.php prikazuje metode potrebne za vezu s ostalim modelima, P.1.*

Slika 4.4. sadrži prikaz metoda potrebnih za spajanje s ostalim modelima te se mogu jasno iščitati veze s ostalim modelima. Metoda *users* kazuje kako objekt modela *Project* pripada više objekata modela *User*, a koristi se za dobivanje svih korisnika koji sudjeluju u radu na određenom projektu. Metoda *tasks* kazuje kako objekt modela *Project* sadrži više objekata modela *Task*, a koristi se za dobivanje svih zadataka vezanih za određeni projekt. Metoda *manager* kazuje kako objekt modela *Project* pripada jednom objektu modela *User*, a koristi se za utvrđivanje voditelja projekta.

Izuzev metoda za vezu modela, *Project* sadrži *mutators* i *accessors* metode prikazane na slici 4.5.

```
    public function getPercentageAttribute()
    {
      $tasksTime = $this->tasks()->whereNotNull('task_id')
                                  ->sum('estimated_time');

      $solvedTasksTime = $this
                           ->tasks()
                         ->whereNotNull('task_id')
                           ->where('completed', true)
                           ->sum('estimated_time');

          if( $solvedTasksTime == 0 )
          {
            return number_format( 0, 2 );
          }
          else
          {
            return number_format(( $solvedTasksTime / $tasksTime )
                                                    * 100, 2 );
          }
    }

    public function getDeadlineAttribute($value)
    {
        $value = date_create($value);
        $value = date_format($value, 'd.m.Y.');
        return $value;
    }

    public function setDeadlineAttribute($value)
    {
        $this->deadline = date('Y-m-d', strtotime(trim($value)));
    }
```

Sl. 4.5. *Isječak koda iz dokumenta ./app/Models/Project.php prikazuje metode accessorsa i mutatorsa, P.1.*

Mutator setDeadlineAttribute neposredno prije spremanja projekta u bazu podataka ispravlja format atributa *deadline* tipa DATE u format koji podržava MySQL baza podataka (YYY-MM-DD). Suprotno njemu *accessor getDeadlineAttribute* ispravlja format datuma koji dobiva iz baze podataka u format DD.MM.YYYY, dok *accessor getPercentageAttribute* računa postotak završenosti projekta.

4.1.3. Task

Model *Task* predstavlja relaciju *tasks* što je definirano atributom *table*. Na slici 4.6. prikazane su metode koje definiraju veze modela *Task* s ostalim modelima.

```
    public function users()
    {
        return $this->belongsToMany('App\Models\User', 'task_user',
                                            'task_id', 'user_id');
    }

    public function project()
    {
      return $this->belongsTo('App\Models\Project', 'project_id');
    }

    public function categoryTasks()
    {
      return $this->hasMany('App\Models\Task', 'task_id', 'id');
    }

    public function category()
    {
        return $this->belongsTo('App\Models\Task', 'task_id', 'id');
    }
```

Sl. 4.6. *Isječak koda iz dokumenta ./app/Models/Task.php prikazuje metode potrebne za vezu s ostalim modelima, P.1.*

Slika 4.6. sadrži prikaz metoda potrebnih za uspostavljanje relacija s ostalim modelima. Metoda *users* kazuje kako objekt modela *Task* pripada više objekata modela *User*, a koristi se za dobivanje svih korisnika koji su izvršitelji zadatka. Metoda *project* kazuje kako objekt modela *Task* pripada jednom objekatu modela *Project*, a koristi se za dobivanje projekta određenog zadatka. Metoda *categoryTasks* kazuje kako objekt modela *Task* ima više objekata modela *Task*, a koristi se za dobivanje zadataka određene skupine zadataka. Metoda *category* kazuje da objekt modela *Task* pripada jednom objektu modela *Task*, a koristi se kako bi se dobila skupina zadataka kojoj pripada određeni zadatak.

Model *Task* također sadrži metode *getDeadlineAttribute* i *setDeadlineAttribute* koje su već pojašnjene.

4.2. Kontroleri

Kontroler djeluje kao sučelje između modela i pogleda, te obrađuje sve zahtjeve korisnika. Kontroler zaprima korisnikove zahtjeve, izvrši potrebne akcije te vraća odgovarajući odgovor pogledu. Unutar kontrolera odvija se logika aplikacije kao što je validacija primljenih HTTP zahtjeva, odnosno poslanih korisnikovih podataka, pozivanje modela radi potrebe obrade podataka te autorizacija zahtijeva. Svi kontroleri nalaze se u direktoriju *./app/Http/Controllers/*, ali cijela logika obrade HTTP zahtjeva za koju su kontroleri i zaduženi nalazi se u *./app/Http/*, a sastoji se od već spomenutih kontrolera, *Requests*, *Middleware* i *Composers*.

Unutar kontrolera odvija se proces validacije podataka. Validacija se može izvršiti na dva načina putem dodavanja novih *Requestova* ili uz pomoć klase *Validator*. Ovaj aplikacija koristi klasu Validator kako bi provjerila podatke koje korisnik pošalje, a sama pravila se nalaze u modelima. Slika 4.7. prikazuje isječak koda u kojem se definira varijabla *validator* koja će poslužiti za provjeru podataka koji se nalaze u varijabli prilikom registracije korisnika.

```
$validator = Validator::make($request->input(), User::$registration_rules);
```

Sl. 4.7. *Definiranje objekta validator klase Validator koja nad podacima koje korisnik predaje provodi pravila registration_rules, ./app/Http/Controllers/AuthController.php, P.1.*

Osim validacije korisniku je nužno onemogućiti pristup određenim URL-ovima (engl. *Uniform Resource Locator*), npr. ako je korisnik prijavljen u aplikaciju on više nema pravo pristupa stranici za prijavu. Logika za kontrolu pristupa se nalazi u *Middlewareu*, a Laravel odmah na početku nudi već gotovo rješenje za autentificirane korisnike i za goste, a te grupe *middleware* povezuju se u *./app/Http/Kernel.php.* U aplikaciji se koriste samo dva *middleware*, a to su *auth* za autentificirane korisnike i *guest* za neautentificirane korisnike.

Autorizacija korisnika za obavljanje određenih zahtjeva odvija se posredstvom klase *Gate*, ali sama autorizacija nalazi se u direktoriju *./app/Policies/*. Unutar direktorija nalaze se dva dokumenta*; ProjectPolicy.php* i *TaskPolicy.php* koji u sebi sadrže metode za autorizaciju korisnika. Na slici 4.8. je prikazana je metoda *kingMethod*, u dokumentu *ProjectPolicy.php* koja služi za davanje ovlasti nad vođenjem projekta samo voditelju projekta.

```
  public function kingMethod(User $user, Project $project)
    {
        return $user->id === $project->manager_id;
    }
```

Sl. 4.8. *Metoda za davanje ovlasti nad projektom ./app/Policies/ProjectPolicy.php, P.1.*

Svi kontroleri i njihove metode koje se trebaju pozvati na traženi URL povezani su preko dokumenta *./app/Http/routes.php*. Unutar tog dokumenta mogu se pronaći svi dostupni URL-ovi te imena kontrolera i metoda za koje su vezani. Na slici 4.9. mogu se vidjeti dvije definirane rute, ruta s parametrom *resource* se veže za kontroler *ProjectController* te sadrži šest (budući da je isključena metoda *index*) URL-ova koji koriste popularni REST (engl. *Representational State Transfer*), odnosno stil dizajniranja web aplikacija u kojem se HTTP koristi za četiri CRUD (engl. *Create, Read, Update, Delete*) operacije. URL-ove koje je ta ruta stvorila moguće je vidjeti na stranicama Laravelove dokumentacije u literaturi [1]. Dok je ruta s metodom *post* povezana s kontrolerom *TaskController* i metodom *checkTask*. Rezultat ove rute je URL */project/{project_id}/task/checkTask* gdje „{project_id}“ predstavlja identifikacijski broj projekta koji se prosljeđuje metodi.

```
Route::resource('project', 'ProjectController',
                  ['except' => ['index']]);

            Route::post('/project/{project_id}/task/checkTask', array(
'uses' => 'TaskController@checkTask' ));
```

Sl. 4.9. *Isječak koda iz dokumenta ./app/Http/routes.php koji rpikazuje definiranje dviju ruta, P.1.*

Metoda *postLogin* kontrolera *AuthController* prikazana na slici 4.10. zadužena je za prijavu korisnika u aplikaciju. Reagira na *post* zahtjev, a kao parametar prima objekt klase *Request,* kao i svaka druga metoda, koji sadrži parametre korisnikova zahtjeva. Prvo se obrađuje vrsta sigurnosne provjere *Throttle* koja omogućuje blokiranje mogućnosti prijave korisnika ako više puta unese netočne podatke, a radi tako što pamti korisničko ime i na IP adresu korisnika. Broj netočnih zahtjeva koje korisnik može poslati te vrijeme zabrane slanja zahtjeva serveru definirano je u *./app/Http/Kernel.php middleware* grupom *api,* te za svakih pet netočnih zahtjeva korisnik je zaključan jednu minutu. Nakon što se obradi *Throttle* provjera obrađuje se validacija podataka, u slučaju pogreške u samoj validaciji korisnik se vraća na stranicu za prijavu s prikazanim pogreškama, a ako validacija prođe putem klase *Auth* provjerava se podudaranje podataka. Ako postoje dani podaci korisnik se prosljeđuje na radnu ploču.

```
public function postLogin(Request $request)
{
	$validator = Validator::make($request->input(), User::$login_rules);

	$throttles = $this->isUsingThrottlesLoginsTrait();

	if( $throttles && $this->hasTooManyLoginAttempts($request) )
	{
		return $this->sendLockoutResponse($request);
	}
	else
	{
		if ($throttles)
		{
			$this->incrementLoginAttempts($request);
		}

		if( $validator->passes() )
		{
			if( Auth::attempt(['email' => $request->input('email'),
'password' => $request->input('password')],
				$request->input('remember_me')) )
			{
				return redirect()->route('dashboard.index');
			}
			else
			{
				return redirect()->back()->withInput()-
>with('message', 'The email and password you entered don\'t match.');
			}
		}
		else
		{
			return route()->back()->withInput()-
>withErrors($validator);
		}
	}
}
```

Sl. 4.10. *Funkcija za prijavu korisnika u ./app/Http/Controllers/AuthController, P.1.*

Funkcija *indeks* kontolera *DashboardController*, prikazana na slici 4.11., je vrlo jednostavna funkcija koja prikazuje korisnikovu radnu ploču sa svim njegovim projektima. U varijablu *projects* spremaju se deset korisnikovih nedovršenih projekta te se dobivaju podaci o voditelju projekta. Zatim se ta varijabla prosljeđuje pogledu.

```
public function index()
{
      $projects = Auth::user()
                              ->projects()
                              ->with('manager')
                              ->where('completed', 0)
                              ->take(10)
                              ->get();

      return view('dashboard.index', array('projects' => $projects));
}
```

Sl. 4.11. *Funkcija indeks zadužena za prikaz radne ploče, P.1.*

Funkcija *show* kontrolera *ProjectController*, prikazana na slici 4.12., zadužena je za dohvat podataka o projektu i prosljeđivanje istih odgovarajućem pogledu. Ovdje se može vidjeti snaga Laravelovog Eloquenta, konkretnije način dohvata iz baze podataka zvan Eager Loading.

```
public function show($id)
{
      $project = Project::with(['users', 'manager', 'tasks' =>
                        function($query){
                        $query->whereNull('task_id')
                              ->with(['categoryTasks' =>
                               function($query){
                                    $query->with('users');
                                    }]);
                              }])
                              ->where('id', $id)
                              ->first();

      $countTasks = $project->getCountOfTasks();
      $solvedTasks = $project->getCountOfSolvedTasks();

      $permission = ( Auth::user()->id === $project->manager->id ) ? true :
false;

      return view('project.showProject', array('project' => $project,
'countTasks' => $countTasks, 'solvedTasks' => $solvedTasks,
                              'activeTasks' => $countTasks - $solvedTasks,
'permission' => $permission));
}
```

Sl. 4.12. *Funckija show koja dohvaća sve podatke vezane za traženi projekt, P.1.*

Dohvaćanjem projekta koji odgovara traženom identifikacijskom broju, dobivaju se relacije *users* što predstavlja članove projekta, *manager* koji predstavlja voditelja projekta i *tasks* odnosno sve zadatke projekta. Dalje u podupitu se dohvaćaju sve skupine zadataka i u još jednom podupitu dohvaćamo sve zadatke pripadajuće skupine. Snagu Laravela možemo vidjeti u drugom podupitu gdje se izbjegava problem N + 1, koji kaže da za svaku skupinu zadataka moramo raditi još jedan upit, Laravel u ovom slučaju radi samo tri upita za dohvaćanje skupina zadataka i svih pripadajućih zadataka što je vidljivo na slici 4.13. Zatim se dohvaća sveukupni broj i broj izvršenih zadataka te se utvrđuje status korisnika, odnosno provjerava se je li korisnik voditelj zadatka te svi se podaci zatim prosljeđuju pogledu.

```
1. select * from `projects` where `id` = ? limit 1

2. select `users`.*, `project_user`.`project_id` as `pivot_project_id`,
   `project_user`.`user_id` as `pivot_user_id` from `users` inner join
   `project_user` on `users`.`id` = `project_user`.`user_id` where
   `project_user`.`project_id` in (?)

3. select * from `users` where `users`.`id` in (?)

4. select * from `tasks` where `tasks`.`project_id` in (?) and `task_id`
   is null

5. select * from `tasks` where `tasks`.`task_id` in (?, ?, ?, ?)

6. select `users`.*, `task_user`.`task_id` as `pivot_task_id`,
   `task_user`.`user_id` as `pivot_user_id` from `users` inner join
   `task_user` on `users`.`id` = `task_user`.`user_id` where
   `task_user`.`task_id` in (?, ?, ?)
```

Sl. 4.13. *Prikaz Laravelovih generiranih SQL upita za funkciju show slike 4.12*

Kao i sve funkcije u *TaskControlleru* se pozivaju isključivo putem AJAX-a (engl. *Asynchronous JavaScript and XML*) o kojem će biti govora kasnije, tako se i funkcija *checkTask,* koja je prikazana na slici 4.14., isto poziva. Zadaća te funkcije je označavanje završenosti zadatka. Prvo se dohvaća zadatak iz baze podataka prema danom identifikacijskom broju. Ako je zadatak nađen i ako njegovo stanje odgovara pretpostavljenom stanju korisnika onda se mijenja stanje zadatka te se šalje šifrirani odgovor o statusu uspješnosti procesa.

```
public function checkTask(Request $request)
{
	$taskId = $request->input('id');
	$checked = (bool) $request->input('checked');

	$task = Task::find($taskId);
	if( $task )
	{
		if( $task->completed === 0 && $checked == 1 )
		{
			$task->completed = 1;
			$task->save();
		}
		else
		{
			$task->completed = 0;
			$task->save();
		}
		return json_encode(array('status' => 1));
	}
	return json_encode(array('status' => 0));
}
```

Sl. 4.14. *Funckija checkTask za promjenu statusa zadatka*

Funkcija *search* kontrolera *ProfileController*, čiji isječak je prikazan na slici 4.15., je također isključivo AJAX zahtjev čija je svrha automatskog dovršavanje korisnikove pretrage (engl. *autocomplete*) drugih korisnika. Nakon pozivanja funkcije i prolaska validacije, pretražuju se atributi *first_name, last_name i username* modela *User* te se vraćaju stavke u relaciji unutar čijih se atributa nalazi dio traženog teksta odnosno sadržaj varijable *search_string*.

```
$this->string = $request->input('string');

	if( isset($this->string) )
	{
		$validator = Validator::make($request->all(), [
			'string' => 'required|alpha_dash',
		]);

		if( $validator->passes() )
		{
			$users = User::where( function($query){
				$query->whereRaw("CONCAT(first_name, ' ',
last_name) LIKE ?", ["%$this->string%"])
					->orWhereRaw("CONCAT(last_name, ' ',
first_name) LIKE ?", ["%$this->string%"])
					->orWhereRaw("username LIKE ?",
["%$this->string%"]);
			})
			->take(10)
			->get();
```

Sl. 4.15. *Prikaz isječka funkcije search kontrolera ProfileController*

4.3. Pogledi

Pogled je jedan od elemenata MVC-a u kojem se proslijeđeni podaci prikazuju korisniku. Jedino ovu komponentu korisnik i vidi te ona predstavlja korisničko sučelje preko kojeg korisnik komunicira s *web* aplikacijom. Unutar pogleda pozivamo sve vanjske dokumente odnosno JavaScript skripte i CSS dokumente. Svi pogledi se nalaze u direktoriju *./resource/views*, a u prilogu P.4.3. mogu se vidjeti slike *web* aplikacije. Korištenjem Bladea možemo vrlo pregledno stvarati poglede, budući da Blade daje mogućnost nasljeđivanja pogleda. U dokumentu . */resources/views/templates/appNav.blade.php* prikazan je obrazac za sve poglede unutar aplikacije i ako želimo da drugi pogled naslijedi ovaj okvir, iskoristimo jednostavnu Bladeovu sintaksu prikazanoj na slici 4.16.

```
@extends('templates.appNav')
@section('content')

@endsection
```

Sl. 4.16. *Prikaz sintakse nasljeđivanja pogleda, [1]*

Svi se pogledi sastoje od HTML elemenata koji su stilizirani CSS dokumentima, a sama dinamičnost stranica dolazi od JavaScripta.

Svi dokumenti CSS-a se mogu pronaći u . */public/css/,* gdje se može vidjeti da su podijeljeni u segmente što je vidljivo i iz samih naziva dokumenta. Svi se CSS dokumenti pozivaju u već spomenutom dokumentu *appNav.blade.php* što je prikazano na slici 4.17.

```
<!-- STYLES -->
<link href="/css/bootstrap-3.3.6-dist/bootstrap.min.css"
rel="stylesheet" />
<link href="/css/sweetalert2.min.css" rel="stylesheet" />
<link href="/css/bootstrap-datepicker3.min.css" rel="stylesheet" />
<link href="/css/dragula.min.css" rel="stylesheet" />
<link href="/css/awesome-bootstrap-checkbox.css" rel="stylesheet" />
<link href="/css/chosen.min.css" rel="stylesheet" />

<!-- ICONS -->
<link href="/font-awesome-4.6.1/css/font-awesome.min.css"
rel="stylesheet" />

<!-- CUSTOM STYLES -->
<link href="/css/custom/main.css" rel="stylesheet" />
<link href="/css/custom/autocomplete.css" rel="stylesheet" />
<link href="/css/custom/project.css" rel="stylesheet" />
<link href="/css/custom/showProject.css" rel="stylesheet" />
```

Sl. 4.17. *Dio koda u dokumentu appNav.blade.php kojim pozivamo sve CSS dokumente, P.1.*

Primjer isječka jednog CSS dokumenta *./public/css/custom/main.css* na slici 4.18. prikazuje definiranje CSS klasa. Važno je napomenuti da to nisu klase prema OOP-u (engl. *Object-oriented programming*) paradigmi programiranja, već jednostavna skupina zadanih atributa za stiliziranje prikaza elemenata. Na slici je vidljivo kako je klasi *avatar-icon* definiran gornji i desni rub debljine 3px neisprekidane linije i zadane boje određeno heksadecimalnim brojem ed2553 te je atributom *border-radius* naznačeno zaobljenje kuteva na elementu. Dok klasa *avatar-icon-l* definira veličinu elementa mjernom jedinicom *em,* koja je jednaka veličini slova „m“.

```
.avatar-icon{
  border-top: 3px solid;
  border-right: 3px solid;
  border-color: #ed2553;
  border-radius: 50%;
}

.avatar-icon-l{
  width: 21em;
  height: 20em;
  margin: auto;
}
```

4.18. *Prikaz klasa u isječku dokumentu ./public/css/custom/main.css, P.1.*

U isječku dokumenta *./public/css/custom/showProject.css* prikazanog na slici 4.19. definirani su stilovi jedinstvenog elementa (engl. *id*), te stanja tog elementa. Jedinstveni element označen nazivom *projectName* poprima prikazana vrijednosti odnosno uklanjaju mu se svojstva pozadine i ruba te dodatna vrijednost *important* označava kako ove vrijednosti atributa imaju najveći prioritet. Stanje *hover* označava stanje u kojem korisnik zadržava kursor iznad elementa, ali ne označava ništa, *active* je onaj trenutak kada korisnik označi element i *focus* se odnosi na element za unos (engl. *input*) elemente kada su oni označeni te je moguće unositi vrijednosti u njih.

```
#projectName,
#projectName:hover,
#projectName:active,
#projectName:focus {
    background: none !important;
    border: none !important;
}
```

Sl. 4.19. *Prikaz definiranje ida projectName te njegovih stanja, P.1.*

Svi JavaScript dokumenti mogu se pronaći u *./public/js* te su također podijeljeni u segmente, ali za razliku od CSS dokumenata praksa je da se JavaScript dokumenti pozivaju na kraju HTML dokumenta jer ako bi ih pozvali na početku mogu značajno usporiti brzinu učitavanja stranice.

Kako je već prethodno navedeno, ova aplikacija koristi AJAX zahtjev. To je JavaScript funkcija koja šalje zahtjev prema serveru i prima odgovor bez potrebe ponovnog učitavanje stranice. Što znači da je moguće mijenjati sadržaj baze podataka i dohvaćati željeni sadržaj u bazi podataka bez novog učitavanja. Ali kako bi ti zahtjevi radili potrebno je konfigurirati postavke, budući da Laravel koristi CSRF (engl. *Cross Side Request Forgery*) zaštitu te na svakom zahtjevu traži skup znakova koje samo prijavljeni korisnik može imati. Konfiguracija je vidljiva na slici 4.20. Dio konfiguracije nalazi se i u *./resources/views/templates/appNav.blade.php* pod *meta* elementu naziva *_token*.

```
// location.protocol gives http or https
// location.hostname gives hostname
var myUrl = location.protocol + "//" + location.hostname + "/";

// ajax setup for csrf tokens
$.ajaxSetup({
   headers: { 'X-CSRF-Token' : $('meta[name=_token]').attr('content') }
});
```

Sl. 4.20. *Prikaz konfiguracije potrebne za provođenje AJAX zahtjeva, P.1.*

Na slici 4.21. prikazan je isječak koda dokumenta *./public/js/custom/showProject.js* čiji je zadatak promijeniti status zadatka nakon što je označen. Prva funkcija reagira na promjenu stanja samo elementa za unos u ovom slučaju *checkboxa,* te dohvaća identifikacijski broj zadatka za koji je vezan element i provjerava je li element označen ili je oznaka uklonjena. Ovisno o stanju elementa poziva se funkcija s odgovarajućim danim statusom. Funkcija koja se poziva je *checkTask* koja za parametre prima identifikacijski broj prethodno označenog zadatka i status zadatka. Funkcija *checkTask* AJAX zahtjevom s zadanim parametrima šalje zahtjev prema serveru te se sa serverske strane poziva već prikazana metoda *checkTask* kontrolera *TaskController* te se dinamički mijenja stanje zadatka u bazi podataka i zatim se status o uspješnosti promjene stanja zadatka šalje natrag, i ako je nastala greška s serverske strane prikaže se pogreška samom korisniku, a ako se sve izvrši u redu korisniku se samo promjeni stanje *checkboxa.*

```
$('div#Tasks-list').on('click', 'input[type="checkbox"]', function(){

	var taskId = $(this).val();
	if( $(this).is(':checked') )
	{
		checkTask(taskId, 1);
	}
	else
	{
		checkTask(taskId, 0);
	}
});

function checkTask( taskId, checked )
{
	$.ajax({
		url: myUrl + 'project/' + project + '/task/checkTask',
		dataType: 'json',
		data: {'id': taskId,
				'checked': checked},
		type: 'post'
	}).done( function(data){
		var responseStatus = data.status;
		switch( responseStatus )
		{
			case 0:
				var systemNotification = noty({
				  text: error_message,
				  layout: 'bottomRight',
				  type: 'error',
				  theme: 'defaultTheme',
				  timeout: 5000,
				});
				break;
			case 3:
				var systemNotification = noty({
				  text: validation_error,
				  layout: 'bottomRight',
				  type: 'error',
				  theme: 'defaultTheme',
				  timeout: 5000,
				 });
				break;
		}
	}).fail( function(data){
		var systemNotification = noty({
		  text: connection_error_message,
		  layout: 'bottomRight',
		  type: 'error',
		  theme: 'defaultTheme',
		  timeout: 5000,
		 });
	});
}
```

Sl. 4.21. *Isječak koda s klijentove strane nužan za promjenu stanja zadatka, P.1.*

5. ZAKLJUČAK

Ovaj rad prikazuje detaljnu izradu web aplikacije zadanim tehnologijama, fokus je stavljen na serverski dio odnosno na *back-end*. Unutar rada je prikazana izrada baze podataka te usporedba klasičnog načina izrade i Laravelovog. Također je i objašnjena struktura web aplikacije prema MVC paradigmi koja je srž Laravelovog razvojnog okruženja, te prema tome i postoje poglavlja modela, kontrolera u pogleda. Danas je razvoj web aplikacija u velikom usponu te i same tehnologije kojima se razvijaju se konstantno mijenjaju i nastaju nove. PHP kao takav je jako star i možda lošiji u nekim aspektima, ali upravo ovakvo razvojno okruženje daje PHP-u sasvim novu definiciju.

LITERATURA

[1] Laravelova dokumentacija, https://laravel.com/ (16.06.2016.)

[2] Laravel 5 Essentials, B. Martin, Packt Publishing, 2015.

[3] Informacije o MVC paradigmi: http://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm (20.06.2016)

[4] Informacije o MVC paradigmi:

http://laravelbook.com/laravel-architecture/ (20.06.2016)

[5] Službene stranice Wikipedija: https://en.wikipedia.org/wiki/Main_Page (22.06.2016.)

[6] Informacije o IoC-u i za sliku 2.3.: https://www.sitepoint.com/dependency-injection-laravels-ioc/ (20.06.2016)

[7] Službene stranice XAMPP-a: https://www.apachefriends.org/index.html (05.04.2016.)

[8] Službene stranice PHP-ove dokumentacije, http://php.net/ (05.04.2016)

[9] Službene stranice W3C, https://www.w3.org/ (05.05.2016.)

[10] Stranice W3Schoolsa, http://www.w3schools.com/ (10.04.2016.)

[11] Službene stranice jQueryeve dokumentacije, https://jquery.com/ (05.04.2016)

[12] Službene stranice Bootstrapove dokumentacije, http://getbootstrap.com/ (10.04.2016.)

[13] Nastavni materijali kolegija Baze podataka, https://loomen.carnet.hr/course/view.php?id=3568 (15.10.2015.)

[14] Službene stranice MySQL-ove dokumentacije: https://www.mysql.com/ (21.06.2016)

[15] Stranice Stackoverflowa, http://stackoverflow.com/ (10.04.2016.)

[16] Cijeli kod aplikacije za vođenje projekata na stranicama GitHuba, https://github.com/jbajic/projectManagement (26.05.2016.)

SAŽETAK

Tema završnog rada izrada *web* aplikacije korištenjem modernih tehnologija. Izrada web aplikacije podijeljena je na više dijelova. U prvom dijelu se bira tehnologija koja će se koristiti tijekom izrade aplikacije, zatim slijedi postavljanje uvjeta i zahtjeva koje baza podataka mora zadovoljavat za web aplikaciju. Nakon što je baza podataka kreirana počinje se s izradom web aplikacije poštujući MVC paradigmu i tako odvajajući dijelove aplikacije u pojedine direktorije. Prvo je nužno postaviti modele pritom poštujući već definirane veze iz baze podataka budući da su modeli odgovorni za rad s bazom podataka. Kontroleri predstavljaju sučelje između modela i pogleda, te primaju zahtjeve od korisnika, procesuiraju te zahtjeve i zatim kreiraju i predaju potrebne podatke pogledu koji samo prezentira te podatke krajnjem korisniku.

Ključne riječi: web, Laravel, MVC, programiranje, aplikacija, PHP, HTML, JavaScript, jQuery, Bootstrap, CSS, MySQL, vođenje projekata, projekt, zadaci

ABSTRACT

LARAVEL'S PROJECT MANAGEMENT APPLICATION

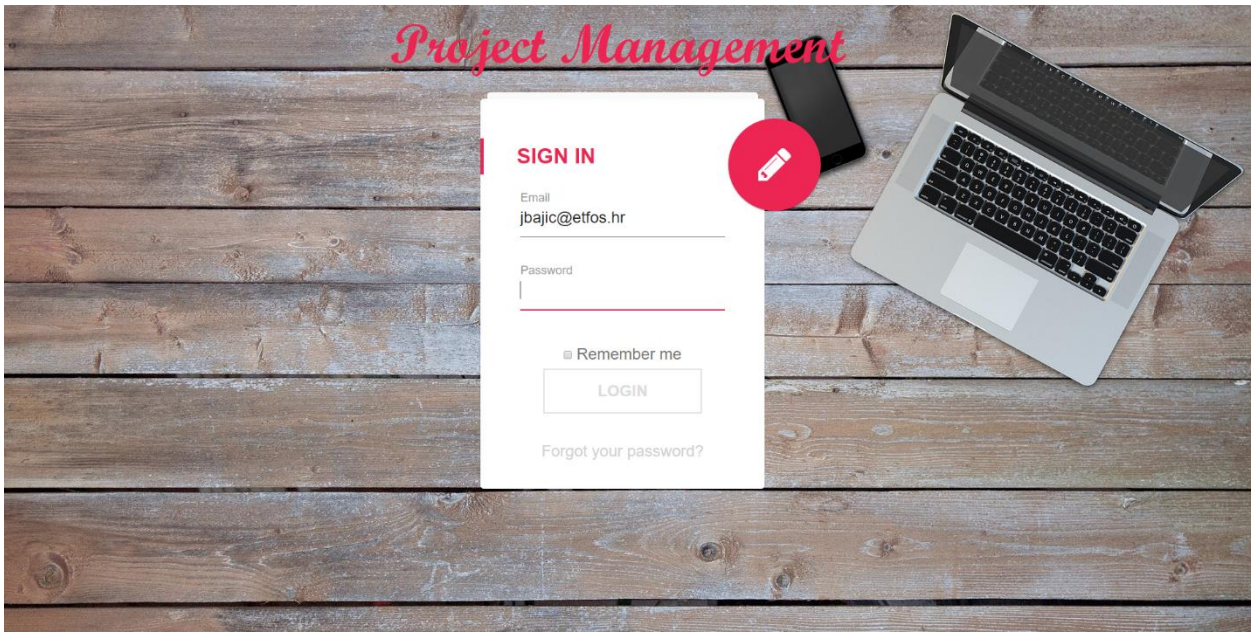
The theme of the final work is development of web application using modern technologies. Creating a web application is divided into several parts. In the first part technologies that will be used during the development of applications are being chosen, followed by setting the conditions and requirements that the database must comply with the web application. Once the database has been created, begins the development of web application respecting the MVC paradigm and thus separating parts of the application into individual directories. It is necessary to set up models while respecting the already defined relationships from the database since the models are responsible for working with the database. Controllers are the interface between the model and the view, after receiving requests from users, they must process those requests, and then submit the required information to the view which is presented to the user.

Key words: web, Laravel, MVC, programming, application, PHP, HTML, JavaScript, jQuery, Bootstrap, CSS, MySQL, project management, project, tasks

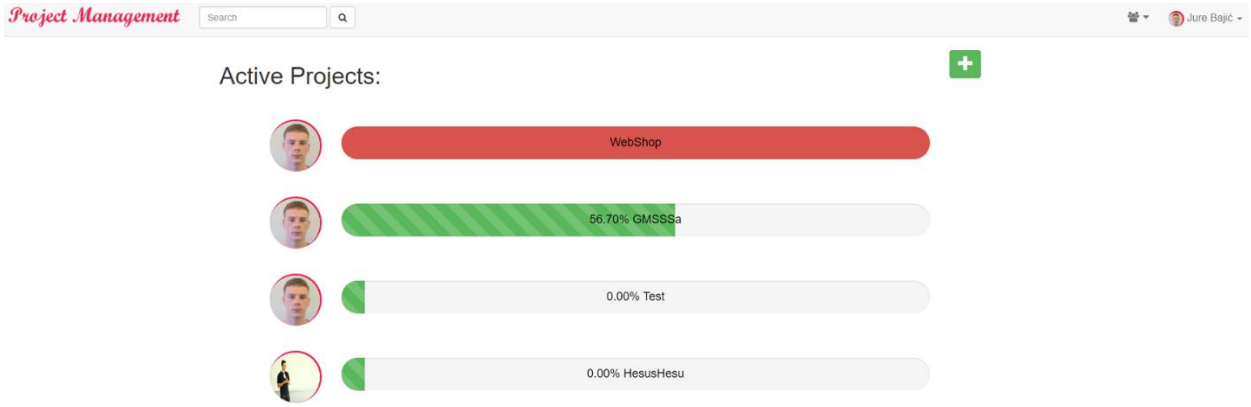
PRILOZI

[1] P. 1. Priložena je web aplikacija u digitalnom formatu i kao literatura

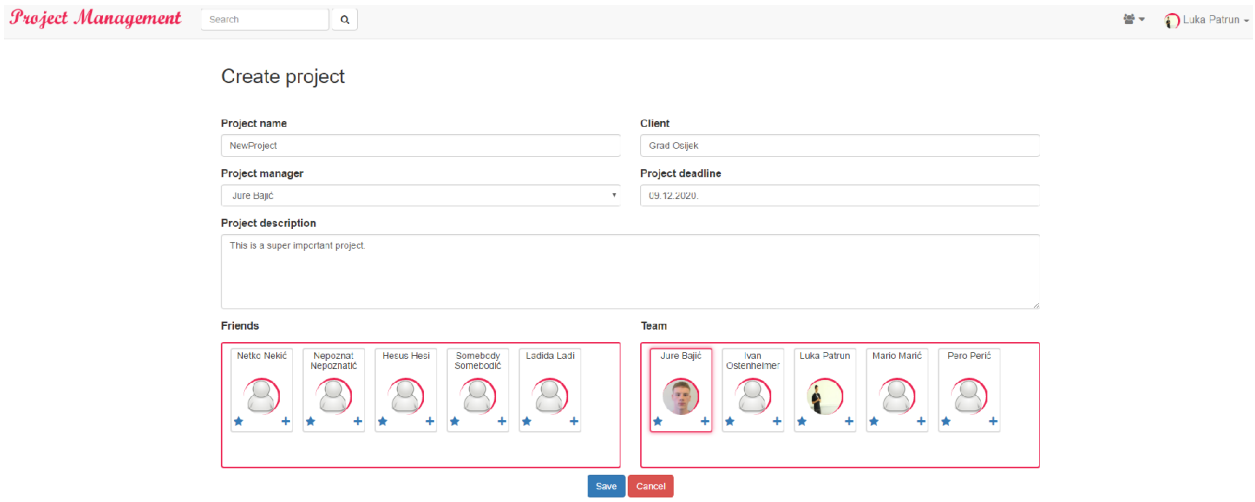
[2] P.4.3. Slike *web* aplikacije



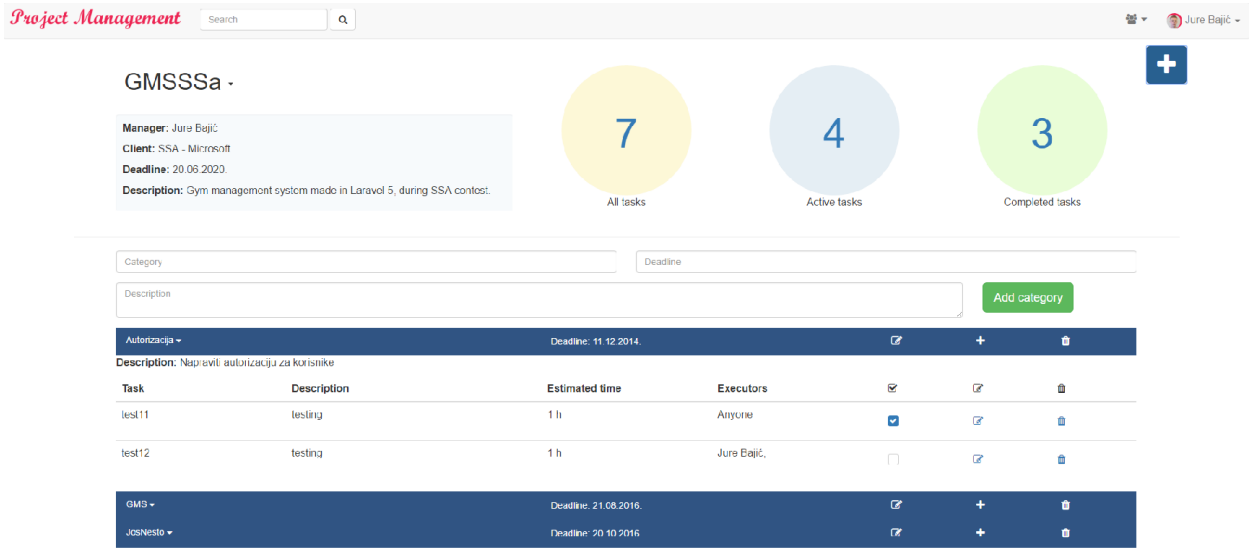
Sl. 4.3.1. *Stranica za prijavu i registraciju korisnika*



Sl. 4.3.2. *Korisnikova radna ploča s prikazanim projektima, njihovim postotkom završenosti, te slikama voditelja projekata*



Sl. 4.3.3. *Stranica za kreiranje novog projekta*



Sl. 4.3.4. *Stranica za upravljanjem projektom, gdje su prikazane sve skupine zadataka i pripadajući zadaci te ostali parametri projekta*

ŽIVOTOPIS

Jure Bajić rođen je 23.8.1994. u Osijeku. Osnovnu školu je pohađao u Višnjevcu. Za to vrijeme aktivno je trenirao ples i vaterpolo. 2009 godine upisuje Prirodoslovno-matematičku gimnaziju, nakon čega upisuje Elektrotehnički fakultet u Osijeku. Na drugoj godini opredijelio se za smjer komunikacije. Na Microsoftovom natjecanju, Software StartUp Academy 2016. odnosi pobjedu s ekipom GymTeam i aplikacijom Palester te osvaja nastup na završnici u Poreču.

U Osijeku, lipanj 2016. Jure Bajić

Potpis: